DOI:
10.1145/3381991.3395610

Document Version
Peer reviewed version

A Data Access Model for Privacy-Preserving Cloud-IoT Architectures

Maribel Fernández
King's College London
United Kingdom
maribel.fernandez@kcl.ac.uk

Alex Franch Tapia
King's College London
United Kingdom
alex@privasee.co.uk

Jenjira Jaimunk
King's College London
United Kingdom
jenjira.jaimunk@kcl.ac.uk

Manuel Martínez Chamorro
King's College London
United Kingdom
manuel@privasee.co.uk

Bhavani Thuraisingham
University of Texas at Dallas
United States of America
bxt043000@utdallas.edu

ABSTRACT

We propose a novel data collection and data sharing model for cloud-IoT architectures with an emphasis on data privacy. This model has been implemented in Privasee, an open source platform for privacy-aware web-application development, which provides a plug-in module to support IoT application development. Privasee uses a cloud-IoT architecture called DataBank. We provide examples and discuss future extensions.

CCS CONCEPTS

• **Security and privacy → Formal security models**; **Access control**; **Privacy-preserving protocols**; *Logic and verification.*

KEYWORDS

Internet of Things, Privacy-Preserving Platform, Data Privacy, Data Collection, Access Control, Privacy Policy.

ACM Reference Format:
Maribel Fernández, Alex Franch Tapia, Jenjira Jaimunk, Manuel Martínez Chamorro, and Bhavani Thuraisingham. 2020. A Data Access Model for Privacy-Preserving Cloud-IoT Architectures. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3381991.3395610

1 INTRODUCTION

Web services and Internet of Things (IoT) applications pose threats to personal privacy due to the large volume of data obtained from users. To comply with current regulations (e.g., GDPR in the EU, or the recommendations of the FTC in the US) that give users control over their data, a number of privacy-preserving architectures have been proposed for cloud-IoT applications. The distinctive characteristic of these platforms is that data privacy is a core feature, following the "privacy-by-design" paradigm. For example, Databox [15] automatically collects data from IoT devices and stores the data in local hubs. Rather than using a central cloud repository, all data processes and access control are executed locally in the box, under the control of the data owner. An alternative solution is to use a centralised cloud platform to store user's data, such as the Personal Data Vault (PDV) proposed by Mun et al. [17], where users have individual secured repositories. Access control lists, trace-audit and rule recommender are used to manage data policies; however, PDV does not include functionalities for users to select and pre-process the data emanating from IoT devices before storage.

DataBank [12] is a hybrid solution: it provides both a local storage (Data Pocket) as well as a central cloud repository, in contrast with Databox and PDV, and it allows users to control data collection from IoT devices and web applications as well as controlling data access. The collected data is filtered in a local hub and transmitted to the central repository according to a user specified data collection policy (collection control). DataBank also controls which services can access the data, using an access control policy.

In this paper we propose an integrated data collection and access control model for hybrid architectures such as DataBank. Specifically, we describe techniques to specify policies to control data collection and data sharing and to implement the local and central repositories. The Data Pocket is a distinctive feature of DataBank, and it is also a critical component as private data stored in this repository needs to be kept secure and accessible. We discuss alternative solutions for the implementation of the Data Pocket, either as local storage under the responsibility of the user, or as a separate cloud storage in the DataBank. The first alternative is suitable for users that have the resources to manage their local storage, while the second is a practical solution for non-technical users. These techniques have been implemented in Privasee, a cloud-IoT development platform based on the DataBank architecture. Privasee offers data owners a secure way to store data, an interface to specify which data they want to upload and how they want to share it, and mechanisms to enforce users' privacy preferences.

Summarising, this paper makes the following contributions to ensure end-to-end privacy preservation (from data collection to access control) in cloud-IoT architectures:

- We propose a data access model that combines data collection and access control features, with a graph-based language to specify policies covering the whole data life cycle. The graph representation of policies facilitates policy visualisation and analysis. We

SACMAT '20, June 10–12, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7568-9/20/06…$15.00
https://doi.org/10.1145/3381991.3395610

demonstrate how queries can be evaluated using policy graphs, and how policy properties can be checked.
- We describe techniques to implement the data pocket (local store) as well as the central repository in the cloud using an encryption mechanism that ensures that only the data owner and no one else (not even Privasee itself), is able to retrieve private data. Alternatively, users could implement the data pocket locally and upload filtered data to the cloud.
- We provide a proof of concept implementation of the proposed architecture and data access model: Privasee.

This paper is organised as follows: Section 2 recalls preliminary notions on cloud-IoT architectures and privacy. Section 3 presents the data access model and a graph representation of policies, used to analyse policies in Section 4. Section 5 discusses implementation techniques. Related works and conclusion are summarised in Section 6 and 7, respectively.

2 BACKGROUND

We recall the main privacy-related notions that will be needed in the rest of the paper.

2.1 Privacy-Preserving Cloud-IoT Architectures

Most web-based and mobile applications collect private data in order to provide better services to users. However, there is a tension between the benefit that sharing personal data can bring to the user and the privacy consequences of unrestricted data collection. Current regulations give users the right to control the data collected by third parties, however, there is no general solution to the technical challenges in building services that enable individuals to share data while retaining control over what is shared and when. Privacy leaks can occur despite anonymisation [22]. Privacy-aware mechanisms such as differential privacy [10], encryption and anonymisation are useful but not sufficient on their own [9].

Recently, several cloud-IoT platforms have been proposed to help application developers design privacy-aware services. Typical cloud-IoT architectures consist of three layers: an object layer (with hardware and software components to manage IoT devices), a middle layer to deal with the cloud storage and management of data, and an application (or services) layer. The object and application layers have a standard set of features in all cloud-IoT architectures, but there is great variety in the definition of the middle layer (number of sub-layers and proposed technologies). The middle layers are in charge of the transfer between the objects in the layer below and the applications in the upper layer, as well as aggregating data and storing it in cloud databases, logging events for auditing purposes, performing data analysis, connecting various data sources, etc. For more details and examples of cloud-IoT architectures, see [1, 2, 8, 13, 20, 23].

Our chosen cloud-IoT architecture, DataBank [12], has four layers as shown in Fig. 1: application layer (which contains interfaces to allow users and services to interact with the DataBank), cloud layer (which manages storage and controls data access; it includes an access control enforcement module, cloud storage, auditing log, privacy-utility mechanism to recommend services to users according to their privacy preferences, and privacy policy), data pocket layer (a local hub of data from IoT devices; it has a memory and micro processing capabilities, to perform light-weight data processing before submitting to the cloud storage, and includes virtual objects representing IoT devices, a module to enforce data collection policies and communication control unit to control the communication amongst virtual objects), and physical objects layer (or sensors layer), which contains IoT devices which communicate with DataBank through drivers and connect via standard protocols.

The distinctive feature of this architecture is that only data that users want to share is stored in the central repository. Applications communicate with DataBank to access stored data, which is managed in accordance with the policies.

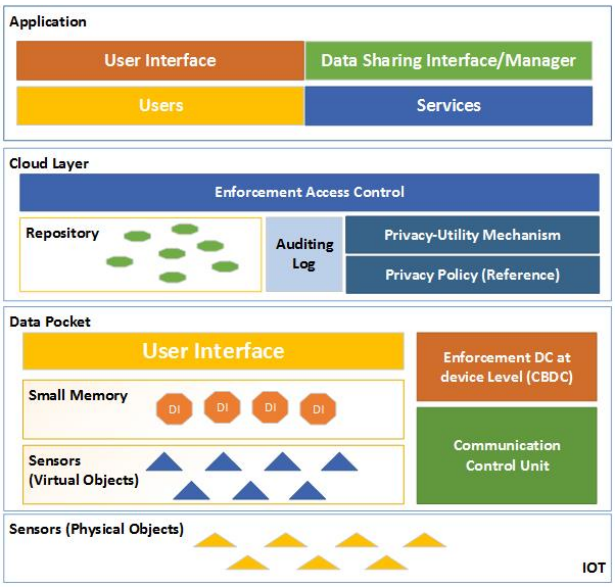


Figure 1: DataBank Architecture

2.2 Category-Based Policies

A variety of access control models and data collection models have been defined, focusing either on the way data collected should be filtered, or the way it should be shared with third parties, see Figure 2. The category-based model [4, 6] is an axiomatic model that has been shown to subsume most of the existing models, and whose formal semantics permits reasoning about policy properties. We recall below the general principles of category-based access control and category-based data collection. Based on these principles, we will define a data access model for cloud-IoT architectures in Section 3.

2.2.1 Category-Based Access Control - CBAC. The CBAC model consists of a set of entities, relationships and axioms (see [6] for more details and examples of category-based access control).

Entities: A countable set $\mathcal{C}$ of categories, denoted $c_0, c_1, \ldots$; a countable set $\mathcal{P}$ of principals, denoted $p_0, p_1, \ldots$; a countable set $\mathcal{A}$ of *actions*, denoted $a_0, a_1, \ldots$; a countable set $\mathcal{R}$ of *resources*, denoted $r_0, r_1, \ldots$; a finite set $\mathcal{A}uth$ of possible *answers* to access requests (e.g., {grant, deny, undetermined}) and a countable set $\mathcal{S}$ of *situational identifiers* to denote environmental information.

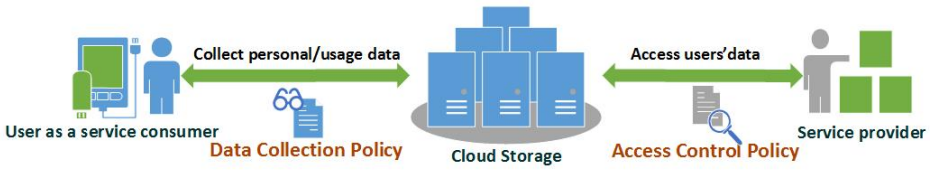


Figure 2: Data Collection and Access Control

Relationships:

- *Principal-category assignment:* $\mathcal{PCA} \subseteq \mathcal{P} \times C$, such that $(p, c) \in \mathcal{PCA}$ iff the principal $p \in \mathcal{P}$ is assigned to the category $c \in C$.
- *Resource-category assignment:* $\mathcal{RCA} \subseteq \mathcal{R} \times C$, such that $(r, c) \in \mathcal{RCA}$ iff the resource $r \in \mathcal{R}$ is assigned to the category $c \in C$.
- *Permission-category assignment:* $\mathcal{ARCA} \subseteq \mathcal{A} \times C \times C$, such that $(a, c_r, c_p) \in \mathcal{ARCA}$ iff action $a \in \mathcal{A}$ on resource category $c_r \in C$ can be performed by the principals in the category $c_p \in C$.
- *Authorisations:* $\mathcal{PAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$, such that $(p, a, r) \in \mathcal{PAR}$ iff the principal $p \in \mathcal{P}$ can perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$.

CBAC relationships must satisfy the following axiom, where $c \subseteq c'$ denotes a hierarchical relation between categories (for example set inclusion or simply equality). A closed-world assumption can be used to derive prohibitions (any action that is not authorised by the policy is forbidden) or additional prohibition relations and axioms can be added (see [6] for more details).

$$(a1) \quad \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, (\exists c_p, c_p', c_r, c_r' \in C, (p, c_p) \in \mathcal{PCA}, (r, c_r) \in \mathcal{RCA} \wedge c_r \subseteq c_r', c_p \subseteq c_p' \wedge (a, c_r', c_p') \in \mathcal{ARCA}) \Leftrightarrow (p, a, r) \in \mathcal{PAR}$$

2.2.2 Category-Based Data Collection - CBDC. Data collection policies specify which data from user devices should be stored. CBDC policies are specified using entities and relationships as described below (we refer to [13] for more details and examples).

Entities: A countable set $\mathcal{D}$ of IoT devices, denoted $d_1, d_2, \ldots$; a countable set $\mathcal{DI}$ of data items, denoted $di_1, di_2, \ldots$; a countable set C of categories, denoted $c_0, c_1, \ldots$; a countable set $\mathcal{A}$ of actions, denoted $a_1, a_2, \ldots$; a countable set $\mathcal{S}$ of services, denoted $s_1, s_2, \ldots$.

Relationships: The following relationships are used to define the authorised actions (additional relations can be included to specify prohibitions).

- *Device Data-Item Assignment: DDIA* $\subseteq \mathcal{D} \times \mathcal{DI}$, such that $(d, di) \in DDIA$ iff the data item di is generated by the device d. Normally, each data item is associated with one device only.
- *Device-Data Item Category Assignment: DDICA* $\subseteq \mathcal{D} \times \mathcal{DI} \times C$, such that $(d, di, c) \in DDICA$ iff the data item $di \in \mathcal{DI}$ generated by the device d is assigned to the category c.
- *Action-Service Category Assignment: ASCA* $\subseteq \mathcal{A} \times \mathcal{S} \times C$, such that $(a, s, c) \in ASCA$ iff action $a \in \mathcal{A}$ of service s can be performed on data items assigned to the category c.
- *Authorisations: ASDID* $\subseteq \mathcal{A} \times \mathcal{S} \times \mathcal{DI} \times \mathcal{D}$, such that $(a, s, di, d) \in ASDID$ iff action $a \in \mathcal{A}$ of service $s \in \mathcal{S}$ is authorised on the data item di generated by $d \in \mathcal{D}$.

CBDC relations must satisfy the following axiom.

$$(dc1) \quad \forall d \in \mathcal{D},\ \forall di \in \mathcal{DI},\ \forall a \in \mathcal{A},\ \forall s \in \mathcal{S}, (\exists c, c' \in C, (d, di) \in DDIA \wedge (d, di, c) \in DDICA \wedge c \subseteq c' \wedge (a, s, c') \in ASCA) \Leftrightarrow (a, s, di, d) \in ASDID$$

3 PRIVACY POLICIES FOR CLOUD-IOT ARCHITECTURE

To specify privacy policies for cloud-IoT architectures, we present a category-based data access (CBDA) model, which combines features of CBDC and CBAC (see Section 2).

3.1 The CBDA Model

Policies in the CBDA model are defined using the sets of entities and relationships of CBDC and CBAC, and additionally, a set of data-sharing categories $\mathcal{DSC}$. These will be used to represent classes of data stored in the central repository. A CBDA policy consists of two parts: a specification of data collection constraints, which indicates how the data collected should be categorised and stored (this can be seen as a CBDC policy, using data-sharing categories instead of services); and a specification of data sharing constraints (this is a CBAC policy, where services are the principals requesting access to data). Data sharing categories are the elements "gluing" both parts of the policy. We now define formally the components of the CBDA model.

Entities: The set of entities of the CBDA model include:

- A countable set $\mathcal{D}$ of sources of data, denoted $d_1, d_2, \ldots$: these are abstractions of data sources and channels; for example, individual sensors, aggregators that combine data from several sensors, clocks, etc.
- A countable set $\mathcal{DI}$ of data items, partitioned into two sets:
  - $\mathcal{D}_{\mathcal{U}}$: unprocessed raw data items, denoted $ud_1, ud_2, \ldots$, these represent data generated by sensors and contextual information (location, time, speed, etc.)
  - $\mathcal{D}_{\mathcal{S}}$: stored data items, denoted $sd_1, sd_2, \ldots$, these represent processed data stored in the central repository, which could be shared with services.
- A countable set $\mathcal{S}$ of services, denoted $s_1, s_2, \ldots$: these represent actual services that process the data.
- A countable set C of categories partitioned into three sets:
  - $C_{\mathcal{D}_{\mathcal{U}}}$: categories of unprocessed data, denoted $udc_0, udc_1, \ldots$; represent categories of data items generated by devices.
  - $C_{\mathcal{D}_{\mathcal{S}}}$: data sharing categories, denoted $dsc_1, dsc_2, \ldots$; these represent categories of data stored in the central repository.
  - $C_{\mathcal{S}}$: service categories, denoted $sc_0, sc_1, \ldots$

- A countable set $\mathcal{A}$ of actions, partitioned into two sets:
  - $\mathcal{A}_{\mathcal{D}}$: data collection actions, denoted $da_1, da_2, \ldots$, are operations on unprocessed data that produce data items ready for storage. We write $(da, ud, sd) \in \mathcal{O}p_d$ if the data collection action da produces sd from ud. Actions of interest in the context of data collection may include upload, average, encrypt, decrypt, etc.
  - $\mathcal{A}_{\mathcal{S}}$: service actions, denoted $sa_1, sa_2, \ldots$, are operations on shared data items stored in the repository, performed by services. Actions of interest in the context of access control may include view data, transfer, share file, etc.

Relationships: The following relations are used to derive the authorised and prohibited actions for data collection and sharing.

- *Device-Data Assignment:* $\mathcal{DUA} \subseteq \mathcal{D} \times \mathcal{D}_{\mathcal{U}}$, such that $(d, ud) \in \mathcal{DUA}$ iff the unprocessed data $ud \in \mathcal{D}_{\mathcal{U}}$ is generated by the data source $d \in \mathcal{D}$. We assume each unprocessed data item is associated with only one data source for simplicity.
- *Data Item-Category Assignment:* $\mathcal{DICA} \subseteq \mathcal{DI} \times C$, which is partitioned into $\mathcal{DICA}_U$ and $\mathcal{DICA}_S$:
  $(ud, udc) \in \mathcal{DICA}_U$ iff the unprocessed data $ud \in \mathcal{D}_{\mathcal{U}}$ is in the unprocessed data category $udc \in C_{\mathcal{D}_{\mathcal{U}}}$.
  $(sd, dsc) \in \mathcal{DICA}_S$ iff the stored data item $sd \in \mathcal{D}_{\mathcal{S}}$ is in the data sharing category $dsc \in C_{\mathcal{D}_{\mathcal{S}}}$.
- *Action-Category Assignment:* $\mathcal{ACA} \subseteq \mathcal{A} \times C \times C$, which is partitioned into $\mathcal{ACA}_D$ and $\mathcal{ACA}_S$:
  $(da, udc, dsc) \in \mathcal{ACA}_D$ iff data collection action $da \in \mathcal{A}_{\mathcal{D}}$ can be performed on data items in the category $udc \in C_{\mathcal{D}_{\mathcal{U}}}$ to produce data in $dsc \in C_{\mathcal{D}_{\mathcal{S}}}$.
  $(sa, dsc, sc) \in \mathcal{ACA}_S$ iff service action $sa \in \mathcal{A}_{\mathcal{S}}$ on data items in data-sharing category $dsc \in C_{\mathcal{D}_{\mathcal{S}}}$ can be performed by services in category $sc \in C_{\mathcal{S}}$.
- *Banned Action-Category Assignment:* $\mathcal{BACA} \subseteq \mathcal{A} \times C \times C$, which is again partitioned into $\mathcal{BACA}_D$ and $\mathcal{BACA}_S$:
  $(da, udc, dsc) \in \mathcal{BACA}_D$ iff data collection action $da \in \mathcal{A}_{\mathcal{D}}$ is banned for data items assigned to the category $udc \in C_{\mathcal{D}_{\mathcal{U}}}$ to produce data in $dsc \in C_{\mathcal{D}_{\mathcal{S}}}$ (we write $(da, udc, \perp)$ if da is forbidden in udc for any dsc);
  $(sa, dsc, sc) \in \mathcal{BACA}_S$ iff service action $sa \in \mathcal{A}_{\mathcal{S}}$ on data items in category $dsc \in C_{\mathcal{D}_{\mathcal{S}}}$ is forbidden for services in category $sc \in C_{\mathcal{S}}$.
- *Service-Category Assignment:* $\mathcal{SCA} \subseteq \mathcal{S} \times C$, such that $(s, sc) \in \mathcal{SCA}$ iff the service $s \in \mathcal{S}$ is assigned to the service category $sc \in C_{\mathcal{S}}$.
- *Authorised Data Collection:* $\mathcal{AD} \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{U}} \times \mathcal{D}_{\mathcal{S}}$ such that $(da, ud, sd) \in \mathcal{AD}$ iff the data collection action $da \in \mathcal{A}_{\mathcal{D}}$ is authorised on unprocessed data $ud \in \mathcal{D}_{\mathcal{U}}$ to produce stored data item $sd \in \mathcal{D}_{\mathcal{S}}$.
- *Prohibited Data Collection:* $\mathcal{BAD} \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{U}} \times \mathcal{D}_{\mathcal{S}}$ such that $(da, ud, sd) \in \mathcal{BAD}$ iff the data collection action $da \in \mathcal{A}_{\mathcal{D}}$ is forbidden on unprocessed data $ud \in \mathcal{D}_{\mathcal{U}}$ to produce stored data item $sd \in \mathcal{D}_{\mathcal{S}}$.
- *Authorised Data Access:* $\mathcal{ADS} \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{S}} \times \mathcal{S}$, such that $(sa, sd, s) \in \mathcal{ADS}$ iff service action $sa \in \mathcal{A}_{\mathcal{S}}$ is authorised on the stored data item $sd \in \mathcal{D}_{\mathcal{S}}$ for the service $s \in \mathcal{S}$.
- *Prohibited Data Access:* $\mathcal{BADS} \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{S}} \times \mathcal{S}$, such that $(sa, sd, s) \in \mathcal{BADS}$ iff service action $sa \in \mathcal{A}_{\mathcal{S}}$ is banned on the stored data item $sd \in \mathcal{D}_{\mathcal{S}}$ for the service $s \in \mathcal{S}$.
- Additional relations $UNDET_D \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{U}} \times \mathcal{D}_{\mathcal{S}}$ and $UNDET_S \subseteq \mathcal{A} \times \mathcal{D}_{\mathcal{S}} \times \mathcal{S}$ contain the tuples (da, ud, sd) (resp. (sa, sd, s)) that are neither authorised nor banned.

Axioms: The axioms specify the way data is collected and stored (axioms $da1$-$da4$), and the way it is accessed by services (axioms $da5$-$da8$). We assume the existence of a hierarchical relation between categories, denoted $c \subseteq c'$ (read c lower than c', i.e., less protected); this can be for example set inclusion or simply equality. Authorised actions are inherited by lower categories from higher categories (axioms $da1$ and $da5$), whereas for prohibitions it is the reverse.

($da1$) $\forall ud \in \mathcal{D}_{\mathcal{U}},\ \forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall da \in \mathcal{A}_{\mathcal{D}},$
$(\exists d \in \mathcal{D}, \exists udc, udc', dsc \in C, (d, ud) \in \mathcal{DUA} \wedge$
$(ud, udc) \in \mathcal{DICA}_U \wedge udc \subseteq udc' \wedge$
$(da, udc', dsc) \in \mathcal{ACA}_D \wedge (da, ud, sd) \in \mathcal{OP}_d \wedge$
$(sd, dsc) \in \mathcal{DICA}_S) \Leftrightarrow (da, ud, sd) \in \mathcal{AD}$

($da2$) $\forall ud \in \mathcal{D}_{\mathcal{U}},\ \forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall da \in \mathcal{A}_{\mathcal{D}},$
$(\exists d \in \mathcal{D}, \exists udc, udc', dsc \in C, (d, ud) \in \mathcal{DUA} \wedge$
$(ud, udc) \in \mathcal{DICA}_U \wedge udc' \subseteq udc \wedge$
$(da, udc', dsc) \in \mathcal{BACA}_D \wedge (da, ud, sd) \in \mathcal{OP}_d \wedge$
$(sd, dsc) \in \mathcal{DICA}_S) \Leftrightarrow (da, ud, sd) \in \mathcal{BAD}$

($da3$) $\forall ud \in \mathcal{D}_{\mathcal{U}},\ \forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall da \in \mathcal{A}_{\mathcal{D}},$
$(\exists d \in \mathcal{D}, (d, ud) \in \mathcal{DUA} \wedge (da, ud, sd) \in \mathcal{OP}_d \wedge$
$(da, ud, sd) \notin \mathcal{AD} \wedge (da, ud, sd) \notin \mathcal{BAD})$
$\Leftrightarrow (da, ud, sd) \in UNDET_D$

($da4$) $\mathcal{AD} \cap \mathcal{BAD} = \emptyset$

($da5$) $\forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall sa \in \mathcal{A}_{\mathcal{S}},\ \forall s \in \mathcal{S},\ (\exists dsc, dsc', sc, sc' \in C,$
$(sd, dsc) \in \mathcal{DICA}_S \wedge dsc \subseteq dsc' \wedge (s, sc) \in \mathcal{SCA} \wedge$
$sc \subseteq sc' \wedge (sa, dsc', sc') \in \mathcal{ACA}_S) \Leftrightarrow (sa, sd, s) \in \mathcal{ADS}$

($da6$) $\forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall sa \in \mathcal{A}_{\mathcal{S}},\ \forall s \in \mathcal{S},\ (\exists dsc, dsc', sc, sc' \in C,$
$(sd, dsc) \in \mathcal{DICA}_S \wedge dsc' \subseteq dsc\ \wedge (s, sc) \in \mathcal{SCA} \wedge$
$sc' \subseteq sc \wedge (sa, dsc', sc') \in \mathcal{BACA}_S)$
$\Leftrightarrow (sa, sd, s) \in \mathcal{BADS}$

($da7$) $\forall sd \in \mathcal{D}_{\mathcal{S}},\ \forall sa \in \mathcal{A}_{\mathcal{S}},\ \forall s \in \mathcal{S}, ((sa, sd, s) \notin \mathcal{ADS} \wedge$
$(sa, sd, s) \notin \mathcal{BADS} \Leftrightarrow (sa, sd, s) \in UNDET_S)$

($da8$) $\mathcal{ADS} \cap \mathcal{BADS} = \emptyset$

Definition 3.1. (CBDA policy) A category-based data access policy is a tuple $\langle \mathcal{E}, \mathcal{R}el \rangle$ where

$$\begin{aligned} \mathcal{E} &= (\mathcal{D}, \mathcal{DI}, \mathcal{A}, C, \mathcal{S}, \subseteq) \\ \mathcal{R}el &= (\mathcal{DUA}, \mathcal{DICA}, \mathcal{SCA}, \mathcal{ACA}, \mathcal{BACA}, \mathcal{AD}, \\ &\quad \mathcal{BAD}, \mathcal{ADS}, \mathcal{BADS}, UNDET) \end{aligned}$$

such that axioms ($da1$)-($da8$) are satisfied.

Axioms ($da1$)-($da8$) can be used to prove properties of policies. They are parametric on the policy relationships, which can be defined by computational rules. A simpler model can be obtained using only axioms ($da1$) and ($da5$) and a closed world assumption: anything that is not explicitly authorised by ($da1$) and ($da5$) is prohibited. Although this simpler model is less expressive, it can be sufficient in many practical scenarios.

Below we describe a graph representation of policies, which we use to control the data collection process and to evaluate data access requests.

3.2 CBDA Policy Graph

Policy graphs [3, 11] are graphs where nodes represent policy entities and edges denote links between entities. Labels are attached to nodes and edges, and store data (in the form of pairs attribute-value) of relevance for the policy.

Definition 3.2 (Record). Let a_i range over a finite set of attribute names, and t_i range over a finite set of values. A record $\mathsf{R} \in \mathcal{REC}$, is a term of the form $\{a_1 = t_1, \ldots, a_n = t_n\}$ where each a_i occurs only once. We use the notation $\mathsf{R}.a$ for attribute selection, and the notation $\mathsf{update}(\mathsf{R}, a, t)$ to modify the value of attribute a in record R to t; if the attribute a does not exist in R, then the field a is added, with value t.

In our representation of policies below, we assume all records have an attribute ent with the name of the entity to which the record belongs.

Example 3.3. The record containing information about the device Security Presence Detector could be specified as follows: R = {ent = *SecurityPresenceDetector*, type = D, component = *camera*}, then R.ent = *SecurityPresenceDetector*, and update(R, function, *OutdoorMotionSensor*) results in the new record {ent = *SecurityPresenceDetector*, type = D, component = *camera*, function = *OutdoorMotionSensor*}.

Definition 3.4 (Policy Graph). A CBDA *policy graph* is a tuple $\mathcal{G} = (V, E, lv, le)$, where V is a set of nodes, E is a set of undirected edges, $lv : V \to \mathcal{REC}$ is a labelling function for nodes, such that, for every $v \in V$, $lv(v).\mathsf{ent} \in \mathcal{E}$ (see Definition 3.1) and $le : E \to \mathcal{REC}$ is a labelling function for edges, such that, for every $e \in E$ between nodes v_1 and v_2, $le(e).\mathsf{adj} = \{v_1, v_2\}$, where $v_1, v_2 \in V$ and $v_1 \neq v_2$.

Node labels contain a field type with values in

$$\{D, D_U, D_S, S, C_{D_U}, C_{D_S}, C_S, A^{\mathfrak{C}}_D, A^{\mathfrak{C}}_S\}$$

such that $lv(v).\mathsf{type} = D$ if $lv(v).\mathsf{ent} = d \in \mathcal{D}$ (that is, D is the type of the nodes representing data sources), and similarly D_U represents unprocessed data, D_S represents stored data items, C_{D_U}, C_{D_S}, C_S categories of unprocessed data, stored data and services resp., S services, and $A^{\mathfrak{C}}_D$, $A^{\mathfrak{C}}_S$ data collection and service actions resp., where the super-index $\mathfrak{C}$ denotes the set of categories to which the action applies (in general, unless it is needed, we will omit this index when referring to types of action nodes). We use the type *DI* when we do not need to distinguish between unprocessed and stored data (i.e., both D_U and D_S are represented by *DI*), and similarly we use the type A to represent both $A^{\mathfrak{C}}_D$ and $A^{\mathfrak{C}}_S$ and the type C to represent C_{D_U}, C_{D_S} and C_S. Nodes of type A include a field *Op* in their label, which specifies the operation represented by the action node.

The type of an edge is determined by the type of its adjacent nodes, that is, $\mathsf{type}(e) = (lv(v_1).\mathsf{type}, lv(v_2).\mathsf{type})$ if $le(e).\mathsf{adj} = \{v_1, v_2\}$.

Since we will represent edge-types (T_1, T_2) as T_1T_2 and the edges are undirected, the types T_1T_2 and T_2T_1 then are not different.

Fig. 3 shows a path in a CBDA policy graph with nine entities (nodes), starting with a node of type D (device) and ending with a node of type S (service). An edge of type DD_U connects a node of type D with a node of type D_U, an edge of type D_UC connects a node of type D_U and a node of type C, etc.

Using types, we can classify nodes into four groups as depicted in Figure 3: *data source nodes* (type D); *data collection nodes*, which include nodes of type D_U (unprocessed data), C_{D_U} (category of unprocessed data), and $A^{\mathfrak{C}}_D$ (actions); *access control nodes* of type D_S (stored data), C_{D_S} (category of data item stored in the central cloud repository), $A^{\mathfrak{C}}_S$ (action on stored data) and C_S (service category); and *data consumer* nodes of type S (service) representing entities that request access to data items.

A *path* of length n in a graph is a sequence $v_0, v_1, \ldots, v_n$ of pairwise distinct nodes, such that, for all $1 \leq i \leq n$, $\{v_{i-1}, v_i\} = le(e).\mathsf{adj}$, for some $e \in E$. In policy graphs, paths of specific types define the authorised and prohibited actions for each kind of data item and service, as shown below.

For edges of type *CC*, we introduce a target attribute to specify the hierarchical relation between adjacent categories. When traversing an edge towards the target the category *increases* (in the opposite direction it *decreases*).

We introduce the notions of constrained and constrained inverse path, and then use these notions to define authorisation and prohibition paths in CBDA policy graphs. In a *constrained path*, edges of type *CC* are traversed towards the target, whereas in a constrained inverse path these edges are traversed in the opposite direction. This is formalised in Definitions 3.5 and 3.6.

Definition 3.5 (Paths in Policy Graphs). A *constrained path* between nodes v_0, v_n in $\mathcal{G}$ is a sequence $v_0, e_1, v_1, e_2 \ldots, e_n, v_n$, where all the nodes are different, such that for all $1 \leq i \leq n$, $le(e_i).\mathsf{adj} = \{v_{i-1}, v_i\}$ and $v_i \in le(e_i).\mathsf{target}$ if target exists in $le(e_i)$.

A *constrained inverse path* between nodes v_0, v_n in $\mathcal{G}$ is a sequence $v_0, e_1, v_1, e_2 \ldots, e_n, v_n$, such that for all $1 \leq i \leq n$, $le(e_i).\mathsf{adj} = \{v_{i-1}, v_i\}$ and $v_{i-1} \in le(e_i).\mathsf{target}$ if target exists in $le(e_i)$.

In a constrained path, categories are traversed from lower to higher and from higher to lower in a constrained inverse path.

Definition 3.6. Let $\mathcal{G}$ be a policy graph and c_1, c_2 be two categories in $\mathcal{C}$. We write $c_1 \subseteq c_2$ if there is a constrained path between the nodes labelled by c_1 and c_2 where all the edges are of type (*CC*). If $c_1 \subseteq c_2$ and $c_2 \subseteq c_1$ then the categories c_1, c_2 are equivalent and we write $c_1 = c_2$.

The concept of a hierarchical relation $\subseteq$ could be applied to other entities such as actions or services.

Definition 3.7 (Types for paths). Let $v_0, \ldots, v_n$ be a path of length n, such that $lv(v_i).\mathsf{type} = T_i$ for $0 \leq i \leq n$. The type of the path is the sequence given by the types of the edges along the path, that is $T_0T_1, T_1T_2, \ldots, T_{n-1}T_n$.

The notation $\mathsf{type}(v_0, v_1, \ldots, v_n) = T_0T_1, T_1T_2, \ldots, T_{n-1}T_n$ will be used to indicate that there is a path $v_0, v_1, \ldots, v_n$ and its edges have types $T_0T_1, T_1T_2, \ldots, T_{n-1}T_n$.

Furthermore, if an edge e between nodes v_i and v_{i+1} in a path $v_0, v_1, \ldots, v_n$ has type *CC*, then we will denote its type as $\overrightarrow{CC}$ if $v_{i+1} \in le(e).\mathsf{target}$, and $\overleftarrow{CC}$ if $v_i \in le(e).\mathsf{target}$ (that is, type $\overrightarrow{CC}$ means that the edge is traversed from a category c_i to a category c_{i+1} where $c_i \subseteq c_{i+1}$, that is, from a lower category to a higher category, and type $\overleftarrow{CC}$ means that the edge is traversed in the opposite direction, from a higher category to a lower one).

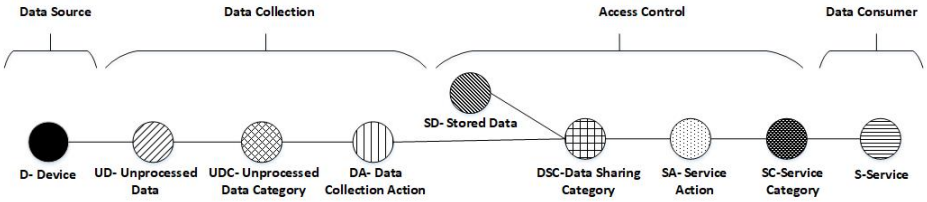


Figure 3: Types of nodes in a CBDA graph

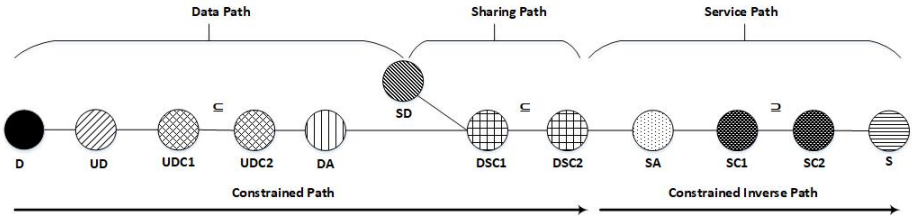


Figure 4: Authorisation Path in a CBDA Policy Graph

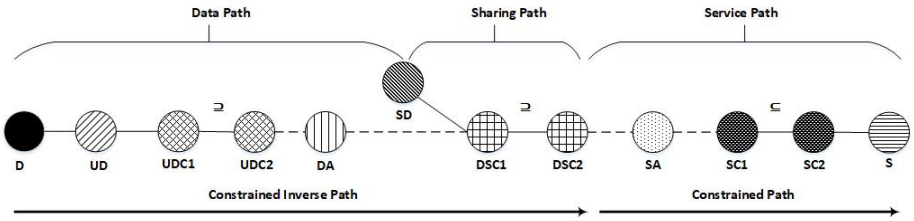


Figure 5: Prohibition Path in a CBDA Policy Graph

Constrained paths between nodes of type C (cf. Definition 3.5) and constrained inverse paths between nodes of type C (cf. Definition 3.5) are characterised by types of the form $(\overrightarrow{CC})^*$ and $(\overleftarrow{CC})^*$ respectively. This characterisation is useful to compute the authorisation and prohibition relations associated with a graph $\mathcal{G}$. As usual, a^* denotes a sequence of the form $\underbrace{a, a, \ldots, a}_{n}$, with $n \geq 0$, so, for example, a path of type $(\overrightarrow{CC})^*$ represents a chain of categories in the $\subseteq$ relation.

Note that an edge may have type $\overrightarrow{CC}$ and also $\overleftarrow{CC}$: if $le(e).\text{target} = \{v_i, v_{i+1}\}$ then the edge can be traversed in both directions and has both types. In fact, a chain of equivalent categories is represented by a path $v_0, v_1, \ldots, v_n$ of both types $(\overrightarrow{CC})^*$ and $(\overleftarrow{CC})^*$, instead of using a cycle $v_0, v_1, \ldots, v_n, v_0$ of type $(\overrightarrow{CC})^*$— our goal is to avoid redundant edges.

Definition 3.8 (Redundant edges). Redundant edges of type DIC (i.e., $D_U C$ or $D_S C$), SC, CC and CA are defined as follows:

- An edge of type DIC, between nodes representing a data item di and a category c, is *redundant* if there is a path of length $n \geq 2$ and type $DIC, (\overrightarrow{CC})^*$ connecting di and c in the graph.
- An edge of type SC, between nodes representing a service s and a category c is *redundant* if there is a path of length $n \geq 2$ and type $SC, (\overrightarrow{CC})^*$ connecting s and c in the graph.
- An edge of type $\overrightarrow{CC}$, between nodes representing two categories c_1 and c_2 is *redundant* if there is a path of length $n \geq 2$ and type $(\overrightarrow{CC})^*$ connecting c_1 and c_2 in the graph.
- An edge of type CA between nodes representing a category c and an action a is *redundant* if there is a path of length $n \geq 2$ and type $(\overrightarrow{CC})^* CA$ connecting c and a in the graph.

The definition of redundant edges takes into account the fact that lower categories inherit actions from higher categories, therefore there is no need to connect actions to lower categories, if there is already a connection through a higher category. In the case of edges of type $\overrightarrow{CC}$, transitive edges are redundant.

Prohibitions. In order to deal with prohibited actions, we consider an extra field auth on labels of edges of type CA and AC, with values in $\{A, B\}$, to represent *authorised* and *banned* actions. By annotating edge types CA and AC with the possible values $\{A, B\}$, we get edges of type CA^A and AC^A (solid edge) and CA^B and AC^B (dotted edge), representing authorisations and prohibitions, respectively. For example, in the policy graph $\mathcal{G}$ shown in Fig. 6, the edge between the category "Modified Footage" and the action "Transfer" is dotted, indicating that this action is not authorised for data items in the

category "Modified Footage", i.e., data emanating from the security presence detector should not be trimmed and transferred to the insurance company.

Note that, as before, we can define redundant edges of type CA (see Definition 3.8), but now we need to consider separately the edges of type CA^A and CA^B: an edge of type CA^A connecting a category c with an action a is redundant if there is already a path of type $(\overrightarrow{CC})^* CA^A$ connecting c and a (this is because lower categories inherit permissions from higher categories); an edge of type CA^B is redundant if there is another path of type $(\overleftarrow{CC})^* CA^B$ connecting the same nodes. This is because higher categories inherit prohibitions from lower categories, so there is no need to add the edge of type CA^B from a higher category to an action if a lower category is already connected to this action.

Definition 3.9 (Authorisation Path). An *authorisation path* in CBDA consists of three sub-paths: a data path, a sharing path and a service path. In each subpath, the edges of type CA must be authorisation edges, i.e., CA^A.

- The data path is a constrained path of type

$$DD_U, D_U C_{D_U}, (\overrightarrow{C_{D_U} C_{D_U}})^*, C_{D_U} A_D^{\mathfrak{C}}, A_D^{\mathfrak{C}} C_{D_S}, C_{D_S} D_S.$$

- The sharing path is a constrained path of type

$$D_S C_{D_S}, (\overrightarrow{C_{D_S} C_{D_S}})^*.$$

- The service path is a constrained inverse path of type

$$C_{D_S} A_S^{\mathfrak{C}}, A_S^{\mathfrak{C}} C_S, (\overleftarrow{C_S C_S})^*, C_S S.$$

According to the definition, the data path starts in a node of type D (data source), which is followed by a node of type D_U (unprocessed data), one or more unprocessed-data categories nodes of type C_{D_U} and a data collection action node of type $A_D^{\mathfrak{C}}$, ending in a stored data node of type D_S. Recall that in a constrained path, edges of type CC are traversed towards the target. The sharing path starts in the end node of the data path, which is followed by C_{D_S} nodes. The service path starts in the end node of the sharing path, which is followed by a service action node of type $A_S^{\mathfrak{C}}$ and one or more service category nodes of type C_S, ending in a service node of type S.

Example 3.10. Fig. 4 depicts an authorisation path that starts with a data path (from D to UD, $UDC1$, $UDC2$, DA, $DSC1$ and SD), then continues with a sharing path (SD to $DSC1$ to $DSC2$), and ends with a service path that ends in a node of type S. Note that in the service path, the edges of type CC are traversed from a higher to a lower category ($\overleftarrow{CC}$), so this is a constrained inverse path.

Definition 3.11 (Prohibition Path). A *prohibition path* in CBDA consists of three sub-paths: data path, sharing path and service path; however, unlike authorisation paths, the edges of type CA must be prohibition edges (i.e., CA^B) and the data and sharing paths must be constrained inverse paths whereas the service path must be a constrained path. More precisely,

- the data path is a constrained *inverse* path of type

$$DD_U, D_U C_{D_U}, (\overleftarrow{C_{D_U} C_{D_U}})^*, C_{D_U} A_D^{\mathfrak{C}}, A_D^{\mathfrak{C}} C_{D_S}, C_{D_S} D_S.$$

- The sharing path is a constrained *inverse* path of type

$$D_S C_{D_S}, (\overleftarrow{C_{D_S} C_{D_S}})^*.$$

- The service path is a constrained path of type

$$C_{D_S} A_S^{\mathfrak{C}}, A_S^{\mathfrak{C}} C_S, (\overrightarrow{C_S C_S})^*, C_S S.$$

According to the definition, the data path is a constrained inverse path (i.e., edges of type CC connecting two categories c_1 and c_2 where $c_1 \subseteq c_2$ are traversed from c_2 to c_1), which starts in a device D and ends in a stored data item node D_S. The sharing path is a constrained inverse path between C_{D_S} nodes starting in the end node of the data path. The service path is a constrained path starting in the end node of the sharing path and ending in a service node S.

Example 3.12. Fig. 5 depicts a prohibition path, starting with a data path and ending with a service path.

We now characterise policy graphs that correspond to CBDA policies, that is, graphs where there is at most one node representing each entity, edge types correspond to classes of entities linked by CBDA relationships, there are no authorised and forbidden actions for the same entities, and there are no redundant edges.

Definition 3.13 (Well-formed Graph). A policy graph (V, E, lv, le) is a *well-formed* if it satisfies the following constraints:

(1) For every $v_1, v_2 \in V$, if $lv(v_1).\text{ent} = lv(v_2).\text{ent}$ and $lv(v_1).\text{type} = lv(v_2).\text{type}$ then $v_1 = v_2$.
(2) Every $e \in E$, where $le(e).\text{adj} = \{v_1, v_2\}$, satisfies one of the following conditions:
   (a) $lv(v_1).\text{type} = D \wedge lv(v_2).\text{type} = D_U$. This corresponds to an edge of type DD_U, which connects a device and an unprocessed data item. Exactly one edge of type DD_U exists for each node of type D_U.
   (b) $lv(v_1).\text{type} = D_U \wedge lv(v_2).\text{type} = C_{D_U}$. This corresponds to an edge of type $D_U C_{D_U}$, which connects an unprocessed data item to an unprocessed data category.
   (c) $lv(v_1).\text{type} = C_{D_U} \wedge lv(v_2).\text{type} = A_D^{\mathfrak{C}} \wedge C_{D_U} \in \mathfrak{C}$. This corresponds to an edge e of type $C_{D_U} A_D^{\mathfrak{C}}$, which connects an unprocessed data category to a data collection action, and in this case $le(e).\text{auth}$ must be defined, such that $le(e).\text{auth} \in \{A, B\}$.
   (d) $lv(v_1).\text{type} = A_D^{\mathfrak{C}} \wedge lv(v_2).\text{type} = C_{D_S} \wedge C_{D_S} \in \mathfrak{C}$. This corresponds to an edge of type $A_D^{\mathfrak{C}} C_{D_S}$, which connects a data collection action and a data sharing category, and in this case $le(e).\text{auth}$ must be defined, such that $le(e).\text{auth} \in \{A, B\}$.
   (e) $lv(v_1).\text{type} = D_S \wedge lv(v_2).\text{type} = C_{D_S}$. This corresponds to an edge of type $D_S C_{D_S}$, which connects a stored data item to a data sharing category.
   (f) $lv(v_1).\text{type} = C_{D_S} \wedge lv(v_2).\text{type} = A_S^{\mathfrak{C}} \wedge C_{D_S} \in \mathfrak{C}$. This corresponds to an edge e of type $C_{D_S} A_S^{\mathfrak{C}}$, which connects a data sharing category to a service action, and in this case $le(e).\text{auth}$ must be defined, such that $le(e).\text{auth} \in \{A, B\}$.
   (g) $lv(v_1).\text{type} = A_S^{\mathfrak{C}} \wedge lv(v_2).\text{type} = C_S \wedge C_S \in \mathfrak{C}$. This corresponds to an edge e of type $A_S^{\mathfrak{C}} C_S$, which connects a service action to a service category, and in this case $le(e).\text{auth}$ must be defined, such that $le(e).\text{auth} \in \{A, B\}$.

(h) $lv(v_1).\text{type} = C_S \wedge lv(v_2).\text{type} = S$. This corresponds to an edge of type $C_S S$, which connects a service category and a service.
(i) $lv(v_1).\text{type} = C \wedge lv(v_2).\text{type} = C \wedge le(e).\text{target} \subseteq \{v_1, v_2\}$, where both types are C_{D_U} or both C_{D_S} or both $C_{\mathcal{S}}$. This corresponds to an edge between categories of unprocessed data, categories of stored data, or categories of services.

Moreover, for every $e_1, e_2 \in E$, if $le(e_1).\text{adj} = le(e_2).\text{adj} = \{v_1, v_2\}$, then either $e_1 = e_2$ or e_1, e_2 have type CA and $le(e_1).\text{auth} \neq le(e_2).\text{auth}$.

(3) If a constrained path and an inverse constrained path start in the same node *ud* of type D_U (resp. *sd* of type D_S, *s* of type S) and end in nodes of type $A^{\mathfrak{C}}_D$ or $A^{\mathfrak{C}}_S$, such that the last edges in the paths are of type CA^A, CA^B respectively, then the end nodes must be different.
(4) There are no redundant edges.

Given a policy graph $\mathcal{G}$, the CBDA relations can be defined in terms of typed paths.

Definition 3.14 (Relations). Let $\mathcal{G}$ be a well-formed policy graph. The following relations are derived from $\mathcal{G}$:

- $\mathcal{DUA}_{\mathcal{G}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}) \mid \text{type}(v_1, v_2) = DD_U\}$.
- $\mathcal{DICA}_{\mathcal{G}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}) \mid \text{type}(v_1, v_2) = DIC\}$. This relation consists of two parts:
  $\mathcal{DICA}_{U_{\mathcal{G}}}$ contains pairs where $\text{type}(v_1, v_2) = D_U C_{D_U}$.
  $\mathcal{DICA}_{S_{\mathcal{G}}}$ contains pairs where $\text{type}(v_1, v_2) = D_S C_{D_S}$.
- $\mathcal{ACA}_{\mathcal{G}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}, lv(v_3).\text{ent}) \mid \text{type}(v_1, v_2, v_3) = AC, CC\}$. This relation consists of two parts:
  $\mathcal{ACA}_{D_{\mathcal{G}}}$, where $\text{type}(v_1, v_2, v_3) = A^{\mathfrak{C}}_D C_{D_U}, C_{D_U} C_{D_S}$.
  $\mathcal{ACA}_{S_{\mathcal{G}}}$ where $\text{type}(v_1, v_2, v_3) = A^{\mathfrak{C}}_S C_{D_S}, C_{D_S} C_S$.
- $\mathcal{BACA}_{\mathcal{G}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}, lv(v_3).\text{ent}) \mid \text{type}(v_1, v_2, v_3) = AC, CC\}$. This relation consists of two parts:
  $\mathcal{BACA}_{D_{\mathcal{G}}}$ where $\text{type}(v_1, v_2, v_3) = A^{\mathfrak{C}}_D C_{D_U}, C_{D_U} C_{D_S}$.
  $\mathcal{BACA}_{S_{\mathcal{G}}}$ where $\text{type}(v_1, v_2, v_3) = A^{\mathfrak{C}}_S C_{D_S}, C_{D_S} C_S$.
- $\mathcal{SCA}_{\mathcal{G}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}) \mid \text{type}(v_1, v_2) = SC_S\}$.
- $\mathcal{AD}_{\mathcal{G}} = \{(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \mid \exists v_0, \exists v_{21}, \ldots, v_{2n}$ s.t. $(v_0, v_1, v_{21}, \ldots, v_{2n}, v_3, v_4, v_5)$ is an authorisation data path and $(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \in \mathcal{OP}_d\}$.
- $\mathcal{BAD}_{\mathcal{G}} = \{(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \mid \exists v_0, \exists v_{21}, \ldots, v_{2n}$ s.t. $(v_0, v_1, v_{21}, \ldots, v_{2n}, v_3, v_4, v_5)$ is a prohibition data path and $(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \in \mathcal{OP}_d\}$.
- $\mathcal{ADS}_{\mathcal{G}} = \{(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \mid \exists v_{21}, \ldots, v_{2n}, v_{41}, \ldots, v_{4n}$ *s.t.*$(v_1, v_{21}, \ldots, v_{2n}, v_3, v_{41}, \ldots, v_{4n}, v_5)$ is an authorisation sharing and service path$\}$.
- $\mathcal{BADS}_{\mathcal{G}} = \{(lv(v_3).\text{ent}, lv(v_1).\text{ent}, lv(v_5).\text{ent}) \mid \exists v_{21}, \ldots, v_{2n}, v_{41}, \ldots, v_{4n}$ *s.t.*$(v_1, v_{21}, \ldots, v_{2n}, v_3, v_{41}, \ldots, v_{4n}, v_5)$ is a prohibition sharing and service path$\}$.
- $UNDET_{D_{\mathcal{G}}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}, lv(v_3).\text{ent}), \mid lv(v_1).\text{type} = A^{\mathfrak{C}}_D, lv(v_2).\text{type} = D_U, lv(v_3).\text{type} = D_S\} - (\mathcal{AD}_{\mathcal{G}} \cup \mathcal{BAD}_{\mathcal{G}})$.
- $UNDET_{S_{\mathcal{G}}} = \{(lv(v_1).\text{ent}, lv(v_2).\text{ent}, lv(v_3).\text{ent}), \mid lv(v_1).\text{type} = A^{\mathfrak{C}}_S, lv(v_2).\text{type} = D_S, lv(v_3).\text{type} = S\} - (\mathcal{ADS}_{\mathcal{G}} \cup \mathcal{BADS}_{\mathcal{G}})$.

The data access policy defined by the graph, denoted by $CBDA_{\mathcal{G}}$, is the tuple $\langle \mathcal{E}_{\mathcal{G}}, \mathcal{DUA}_{\mathcal{G}}, \mathcal{DICA}_{\mathcal{G}}, \mathcal{SCA}_{\mathcal{G}}, \mathcal{ACA}_{\mathcal{G}}, \mathcal{BACA}_{\mathcal{G}}, \mathcal{AD}_{\mathcal{G}}, \mathcal{BAD}_{\mathcal{G}}, \mathcal{ADS}_{\mathcal{G}}, \mathcal{BADS}_{\mathcal{G}}, UNDET_{\mathcal{G}} \rangle$ where $\mathcal{E}_{\mathcal{G}} = (\mathcal{D}_{\mathcal{G}}, \mathcal{DI}_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}}, \mathcal{C}_{\mathcal{G}}, \mathcal{S}_{\mathcal{G}}, \subseteq)$ consists of the sets of data sources, data items, actions, categories, services, and the hierarchical relation $\subseteq$ obtained by computing paths of type CC^* as indicated in Definition 3.6. For example, $\mathcal{D}_{\mathcal{G}} = \{lv(v).\text{ent} \mid lv(v).\text{type} = D, v \in V\}$. The other sets in $\mathcal{E}_{\mathcal{G}}$ are computed similarly.

Example 3.15 (Policy Graph). Fig. 6 shows a CBDA policy graph for a smart home; full edges have type CA^A and dotted edges have type CA^B. Different node types are depicted using different patterns. For example, the black node labelled Security Presence Detector has type D (device); the node Security Footage has type D_U (unprocessed data); the nodes Restricted and Protected represent categories and have type C_{D_U}; Live Stream and Trim have type $A^{\mathfrak{C}}_D$ (data collection actions); Live footage and Trimmed file have type D_S (stored data); Streamed Footage, Trimmed Footage and Modified Footage have type C_{D_S} (they represent data-sharing categories); Live Watch, View File, Share, Sell, Market and Transfer have type $\mathcal{A}_{\mathcal{S}}$ (service actions), the nodes House Owner, Security, Health and Insurance are service categories and have type C_S; and Louis, Metropolitan Police and Royal Oak have type S (services).

Two authorisation paths are highlighted in boldface in Fig. 6. The top one connects the nodes Security Presence Detector and Security Footage with an edge of type DD_U, then continues with an edge of type $D_U C_{D_U}$ (from Security Footage to the node representing the Restricted category), followed by an edge of type $C_{D_U} C_{D_U}$ (between the Restricted and Protected categories), $C_{D_U} A^{\mathfrak{C}}_D$ (between the Protected category and the action Trim), $A^{\mathfrak{C}}_D C_{D_S}$ (between the action Trim and Trimmed Footage category), $C_{D_S} A^{\mathfrak{C}}_S$ (between Trimmed Footage and the service action View File), $A^{\mathfrak{C}}_S C_S$ (between the View File and the service category Security), and $C_S S$ (between the Security category and the service Metropolitan Police).

The graph also shows that the Security Footage from Security Presence Detector is stored as Streamed Footage and will be live watched by the House Owner (authorised action) but will not be transferred to the Insurance service as it is in the Modified Footage category (prohibited action), shown by a dotted edge of type CA^B.

There are also prohibition paths for data generated by the smart meter in Fig. 6: the banned edge connecting the category Confidential and action Upload indicates that the electricity consumption data cannot be uploaded as raw data. If instead we had a full edge between the Confidential category and the action Upload, the raw data could be stored as Raw Consumption but the dotted edge between the data sharing category Raw Utilities and service action View Raw Data means that neither the smart meter supplier nor the house owner is able to view the raw electricity consumption. Replacing the dotted edge (CA^B) between Raw Utilities and View Raw Data with a full edge (CA^A) would permit the House Owner to see the Raw Consumption data, but not the Smart Meter Supplier (dotted edge between View Raw Data and Smart Meter Supplier).

Proposition 3.16. *Let $\mathcal{G}$ be a well-formed policy graph. Then the tuple $CBDA_{\mathcal{G}}$ is a CBDA policy.*

Proof. The relations derived from the graph satisfy the axioms (da_1)-($da8$): The definition of authorisation and prohibition paths reflects axioms ($da1$), ($da2$), ($da5$) and ($da6$); axioms ($da4$) and ($da8$) are satisfied by definition of well-formed graph; and axioms ($da3$) and ($da7$) are satisfied by definition of $UNDET_{D_{\mathcal{G}}}$ and $UNDET_{S_{\mathcal{G}}}$. □

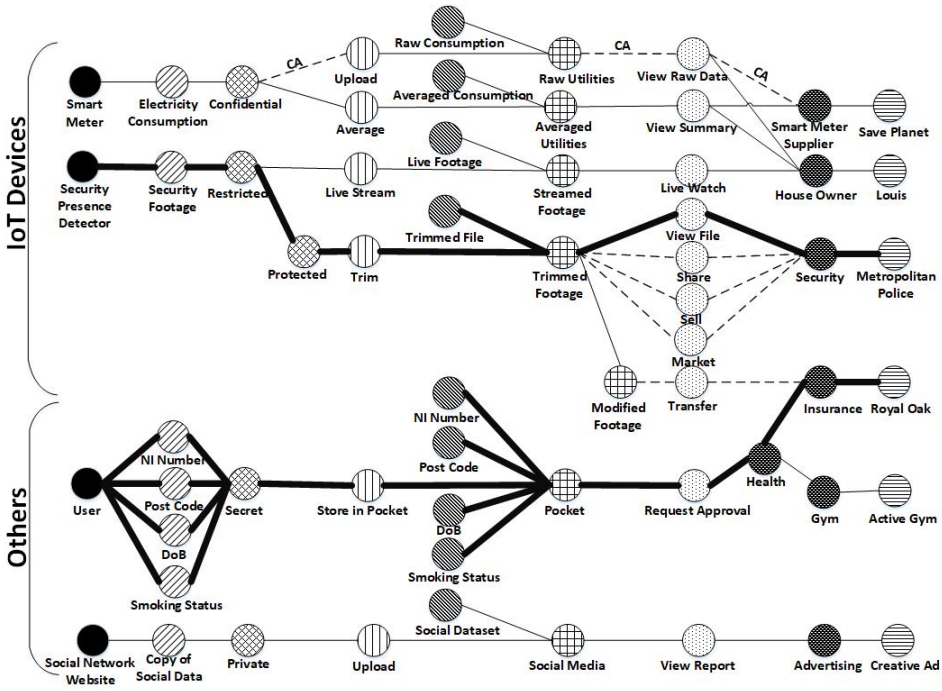


Figure 6: CBDA Policy Graph

Proposition 3.17. *Let $\mathcal{P}$ be a CBDA policy. There exists a well-formed policy graph $\mathcal{G}$ that represents the policy $\mathcal{P}$.*

Proof. The set of nodes in $\mathcal{G}$ is determined by the set $\mathcal{E}$ of entities in the CBDA policy, and similarly, relations in the policy directly map to edges in the graph. The only difficulty is the representation of actions, where we need to establish the set of categories to which they apply and identify actions that have the same name and apply to the same categories (since a well formed graph cannot have multiple nodes of the same name and type). □

4 POLICY ANALYSIS

Meta-data queries, policy content queries, and policy effect queries are traditionally used to analyse policies [5]. This paper focuses on policy content and policy effect queries. The first are used to examine the content of policies, and the latter to check the authorisations and prohibitions specified by the policy.

4.1 Policy content queries

The following are typical queries about policy components (we describe some distinctive cases, others are treated similarly and omitted due to lack of space).

Q1: Are all the data items associated with at least one category? E.g., are all the unprocessed data items associated with at least one unprocessed data category and are all the stored data items associated with at least one data sharing category?

Q2: Are there (permitted or prohibited) actions available for each category (e.g., data collection actions for unprocessed data categories and data sharing categories, service actions for data sharing categories and service categories)? And more precisely, which actions are associated with each category?

Q3: For a given category, what are the associated data? E.g., which unprocessed data belong to a given unprocessed data category and which stored data are in a given data sharing category?

Q4: For a given data item, what actions are permitted? E.g. what data collection actions are permitted for an unprocessed data item? What service actions are permitted for a given stored data item?

Graph-theoretic methods answer all the queries mentioned above when using graph policies as follows:

Q1. All the data items are associated with at least one category if and only if the degree of every node of type D_U is greater than 1 and the degree of every node of type D_S is greater than 0. This is because Def. 3.13 specifies that there is one edge of type DD_U for each node of type D_U, and nodes of type C_{D_S} can only be connected to nodes of type D_S.

Q2. All the categories have some associated (permitted or prohibited) actions if and only if

(1) for each node v of type C_{D_U} there is a path of type $(\overrightarrow{C_{D_U}C_{D_U}})^*, C_{D_U}A_D^{\mathfrak{C}\ A}$ or a path of type $(\overleftarrow{C_{D_U}C_{D_U}})^*, C_{D_U}A_D^{\mathfrak{C}\ B}$ starting in v.
(2) for each node v of type C_{D_S} there is a path of type $A_D^{\mathfrak{C}}C_{D_S}^A, (\overrightarrow{C_{D_S}C_{D_S}})^*$ or a path of type $A_D^{\mathfrak{C}}C_{D_S}^B, (\overleftarrow{C_{D_S}C_{D_S}})^*$ ending in v, as well as a path of type $(\overrightarrow{C_{D_S}C_{D_S}})^*, C_{D_S}A_S^{\mathfrak{C}\ A}$ or a path of type $(\overleftarrow{C_{D_S}C_{D_S}})^*, C_{D_S}A_S^{\mathfrak{C}\ B}$ starting in v.
(3) for each node v of type C_S there is a path of type $A_S^{\mathfrak{C}}C_S{}^A, (\overrightarrow{C_SC_S})^*$ or a path of type $A_S^{\mathfrak{C}}C_S{}^B, (\overleftarrow{C_SC_S})^*$ ending in v.

Q3. To retrieve the set of data items that belong to a category $udc \in C_{D_U}$ or $dsc \in C_{D_S}$:

(1) compute the set $\{v_1, \ldots, v_n\}$ of nodes of type D_U such that for each v_i there is a path of type $D_U C_{D_U}, (\overrightarrow{C_{D_U} C_{D_U}})^*$ starting from v_i and ending in the node of type C_{D_U} representing udc, and output $lv(v_i)$.ent $(1 \leq i \leq n)$.
(2) compute the set $\{v_1, \ldots, v_n\}$ of nodes of type D_S such that for each v_i there is a path of type $D_S C_{D_S}, (\overrightarrow{C_{D_S} C_{D_S}})^*$ starting from v_i and ending in the node of type C_{D_S} representing dsc, and output $lv(v_i)$.ent $(1 \leq i \leq n)$.

Q4. To obtain the set of permitted actions on a given data item represented by a node v of type DI, we distinguish two cases:

(1) If v of type D_U (unprocessed data item), permitted data collection actions are obtained by computing all the paths of type $D_U C_{D_U}, (\overrightarrow{C_{D_U} C_{D_U}})^*, C_{D_U} A_D^{\mathfrak{C}\,A}$ starting at v.
(2) If v of type D_S (stored data item), permitted data collection actions are obtained by computing all the paths of type $A_D^{\mathfrak{C}} C_{D_S}^A, (\overrightarrow{C_{D_S} C_{D_S}})^*, C_{D_S} D_S$ ending at v; and permitted service actions are obtained by computing all the paths of type $D_S C_{D_S}, (\overrightarrow{C_{D_S} C_{D_S}})^*, C_{D_S} A_S^{\mathfrak{C}\,A}$ starting at v.

The complexity associated to each query is polynomial on the size of the policy (more precisely, on the size of the traversed subgraph).

4.2 Policy effect queries

In this section we discuss three policy-effect queries: totality, consistency and absence of conflict. Totality guarantees the policy covers all relevant actions and services for each data item, consistency guarantees no authorisation/prohibition clashes arise from the policy, absence of conflict deals with mutually exclusive actions. We show how to check these properties in general, independently of the categorisation methods used in the CBDA policy.

Definition 4.1 (Totality). A CBDA policy is total if it specifies all authorised and banned actions associated with each service s for every data source d generating data items di.

Definition 4.2 (Consistency). A CBDA policy is consistent if it defines non-contradictory options for any given data item di, i.e., all relevant actions are either permitted or prohibited, but not both.

Assuming there is a well-formed policy graph $\mathcal{G}$ representing the CBDA policy, totality can be verified by computing the relations $\mathcal{AD}_{\mathcal{G}}$, $\mathcal{BAD}_{\mathcal{G}}$, $\mathcal{ADS}_{\mathcal{G}}$ and $\mathcal{BADS}_{\mathcal{G}}$:

Proposition 4.3 (Totality). *A CBDA policy defined by a policy graph $\mathcal{G}$ is total if and only if*

(1) for all nodes ud of type D_U, da of type $A_D^{\mathfrak{C}}$ and sd of type D_S such that $(ud, da, sd) \in Op_d$, $(da, ud, sd) \in \mathcal{AD}_{\mathcal{G}} \cup \mathcal{BAD}_{\mathcal{G}}$; and
(2) for all nodes sd of type D_S and s of type S such that sa applies to sd, $(sa, sd, s) \in \mathcal{ADS}_{\mathcal{G}} \cup \mathcal{BADS}_{\mathcal{G}}$.

Consistency is enforced by axioms ($da4$) and ($da8$), that is, the policy graph satisfies $\mathcal{AD}_{\mathcal{G}} \cap \mathcal{BAD}_{\mathcal{G}} = \emptyset$ and $\mathcal{ADS}_{\mathcal{G}} \cap \mathcal{BADS}_{\mathcal{G}} = \emptyset$, which means that data access policies defined by well formed graphs are consistent by construction.

Absence of conflict means that two mutually exclusive actions on the same data item are not permitted. If an action $da_1 \in A_D^{\mathfrak{C}}$ that produces a stored data item $sd_1 \in D_S$ from an unprocessed data item $ud \in D_U$ is in conflict with an action $da_2 \in A_D^{\mathfrak{C}}$ then the policy should ensure that if da_1 is authorised then da_2 is forbidden and vice versa. Similarly, if an action $sa_1 \in A_S^{\mathfrak{C}}$ performed by a service s_1 on a stored data item $sd \in D_S$ is in conflict with an action $sa_2 \in A_S^{\mathfrak{C}}$ performed by s_2 then the policy should ensure that if sa_1 is authorised then sa_2 is forbidden and vice versa. This kind of checks can be done by computing paths:

Proposition 4.4 (Absence of Conflict). *Let $\mathcal{G}$ be a well-formed policy graph. Assume two actions da_1, da_2 of type $A_D^{\mathfrak{C}}$ (resp. two actions sa_1, sa_2 of type $A_S^{\mathfrak{C}}$) are mutually exclusive.*

The policy graph $\mathcal{G}$ ensures absence of conflict between da_1 and da_2 if for each node ud of type D_U, the set of authorisation paths that start in ud does not contain paths via da_1 and paths via da_2

The policy graph $\mathcal{G}$ ensures absence of conflict between sa_1 and sa_2 if for each node sd of type D_S, the set of authorisation paths that start in sd does not contain paths via sa_1 and paths via sa_2

5 IMPLEMENTATION

In this section we discuss techniques to implement DataBank's repositories. For users with technical resources to manage local storage (e.g., companies that need to protect their data) the Data Pocket is implemented locally. For non-technical users, we propose a solution that does not require installing purpose-built hardware or building a local control hub. Specifically, we propose to implement both the Data Pocket and central repository in the cloud using a novel encryption technique (see Sections 5.1 and 5.2) that ensures that data is kept secure and can only be accessed with owner's authorisation. Having the data pocket in the cloud means that users must upload data to the cloud, even if they do not want to share it at all, but the risks are balanced by the benefits of not having to buy any additional components to implement their own secure storage.

5.1 Proxy Re-Encryption (PRE)

Proxy Re-Encryption (PRE) is a type of Public Key Encryption (PKE) which transforms the ciphertexts under the public key of Alice into ciphertexts decryptable by Bob [18]. It uses a proxy which has a re-encryption key which in conjunction with the ciphertext can modify it to make it decryptable by Bob. The proxy must be able to do this while learning nothing about the underlying message which is encrypted. A PRE environment has three actors [18]: Delegator (delegates decryption rights using re-encryption, i.e., Alice), Delegatee (is granted the right to decrypt ciphertexts, i.e., Bob) and Proxy (responsible for the re-encryption). In the re-encryption process, the proxy transforms ciphertexts under the delegator's public key into ciphertexts that the delegatee can decrypt with his private key.

The simplest use case of a PRE scheme uses a PKE scheme: Alice generates a message and encrypts it with her public key, later she decides to share this with Bob and asks the proxy to re-encrypt it with a re-encryption key. Bob can then retrieve the re-encrypted ciphertext and decrypt it with his private key. Unlike PKE, PRE allows Alice to encrypt the ciphertext without knowing she wanted to share information with Bob. Alice could do this repeatedly for multiple people without having to decrypt her data.

5.2 Umbral: A Threshold PRE Scheme

Umbral is a threshold Proxy Re-Encryption Scheme based on elliptic curve cryptography, which delegates decryption rights using N number of semi-trusted proxies [19]. It is a threshold scheme because it requires a minimum number of proxies to cooperate to perform re-encryption. Umbral improves upon the BBS98 Scheme [7] by adding the properties of unidirectionality and non-interactivity. It is a Hybrid Proxy Re-Encryption scheme using Symmetric Cryptography to encrypt plaintext and Public Key Cryptography to exchange the symmetric key. Re-encryption is achieved by the use of a Key Encapsulation Mechanism (KEM). Umbral encrypts the message with a symmetric key derived from Alice's public key and then generates a ciphertext alongside a capsule or KEM ciphertext. This capsule "encapsulates" or "contains information critical to the computation of the symmetric key". What is re-encrypted is not the ciphertext but the capsule, which encapsulates the symmetric key, thus re-encapsulating the key. The combination of the encapsulation, decapsulation and re-encapsulation provides the functionality of a Proxy Re-Encryption Scheme.

Unlike other PRE schemes, Umbral encrypts the plaintext with an authenticated symmetric encryption algorithm (Chacha20-poly1305 in the PyUmbral implementation [19]). It then creates a capsule that encapsulates the symmetric key. Then Alice generates re-encryption key fragments which are distributed to semi-trusted proxies. These proxies can then use the original capsule and the key fragments to compute capsule fragments to be sent to Bob. These capsule fragments alongside the ciphertext can be sent over non-trusted environments. Bob can then fetch the capsule fragments. Whenever he has a threshold number, he can decapsulate the capsule with his private key. Now, Bob can compute the symmetric key Alice used for encryption. Then he can decrypt the data.

The ciphertext is protected by standard symmetric encryption, and the capsule is protected against forgery by the discrete logarithm problem, allowing it to be safely sent across an unknown network. Furthermore, for every re-encryption key, a shared secret is generated between the delegate's pair and a temporary pair. This makes the scheme unidirectional as the same re-encryption key cannot be used in the other direction. Moreover, given that it uses the delegatee's public key to create the re-encryption key the scheme is non-interactive. It is also non-transitive and single use.

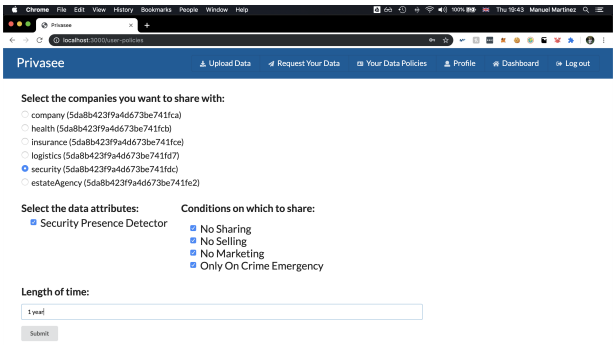
This scheme has the required properties for our application. The use of Symmetric Cryptography in Umbral solves the efficiency problem of PKE schemes in practical applications. Moreover, Umbral has extensive documentation and is fully implemented as a cryptographic library with many helper methods. Note that the number of cloud providers does not change the reliability of the encryption. We only need one cloud provider to hold the ciphertext for any number of receivers. The key issues are ensuring the threshold number of proxies are available and ensuring they do not collude. Ideally, proxies should be in a network of nodes with an incentive not to collude; in our implementation this is achieved by keeping the proxies within our domain.

5.3 Examples in Privasee

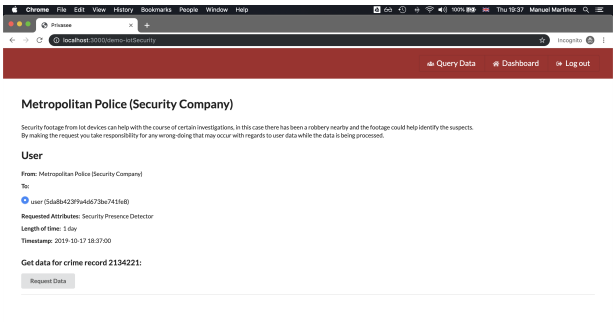
We discuss two scenarios to illustrate how users can specify data collection restrictions, and how data is shared with services.

Example 5.1. To set up a policy, Privasee asks the user a small number of privacy preferences, the data is then shared in accordance with the policy. After signing in, the user selects the services to share information with, data items and categories. For example in Fig. 7a, the user is happy to share security footage from the security presence detector for a year with the Security services with four conditions: no sharing, no selling, no marketing (forbidden actions) and only view the footage file in case of crime emergency (authorised action). The corresponding authorisation and prohibition paths are shown in Fig. 6 (with bold and dashed edges respectively). In Fig. 7b, the metropolitan police (in the security category) requests data from security presence detector for a day, the footage data is automatically shared with this service according to the policy.

Example 5.2. If a user does not specify any data collection restrictions, the data is assigned by default to the Pocket data sharing category, as shown in Fig. 6 (lower bold face path). Data in this category is not shared: any access request by a service will be referred to the user for a decision to be made. The user can create a policy to specify authorisations (or prohibitions) for specific data items and services, or leave the data in the Pocket category for future review. When Royal Oak, an insurance company, requests to access user's NI number, post code, date of birth and smoking behaviour, Privasee informs the user, who has to manually accept or reject this request since no policy has been defined.



(a) User's policy for security presence detector



(b) Data request for security presence detector

6 RELATED WORK

Several approaches for cloud-IoT development combine layered architectures with access control (e.g., using ACL and ABAC policies in [2]) but few focus on management of personal data. The DataBank architecture [12] allows the user to control how data is collected and how it is shared. Here we have proposed an integrated data collection and access control model, CBDA, specifically designed for privacy-preserving cloud-IoT architectures. It covers the whole data cycle, from the point data is generated to the point where it is used. Data collection and access control are modelled separately in CBDA to ensure services never access raw data. Alternatively, the cloud could have been modelled as a service provider (in its own category) but then we would need to allow this service to access raw data. We followed the separation of concerns principle, separating unprocessed and processed data categories, and separating service providers from the cloud provider.

For the definition of policies we use the category-based approach [4], which is suitable for highly dynamic scenarios as required in cloud-IoT applications. A role is a particular case of category, and categories can also be defined on the basis of user, object or environment attributes, hence CBDA subsumes the popular RBAC [21] and ABAC models [14, 16]. The graph-based policy representations we define generalise previous approaches [3, 11] by including data sharing categories as a mechanism to link data and services.

Other cloud-IoT architectures have been proposed (see [1] for an overview), and the data access model and implementation techniques proposed in this paper could be adapted for them. Among the approaches proposed to tackle privacy issues for personal data in cloud-IoT platforms, two are close to ours: Haddadi et al. [15] Databox, and Mun et al. [17] Personal Data Vault. However, these approaches do not provide a language to visualise policies.

Databox [15] is a trusted platform enabling people to manage personal data in a controlled way. Our work is complementary, in that we propose an architecture with a mechanism for users to specify data collection policies at device level as well as data sharing policies for cloud-stored data.

Personal Data Vault [17] offers a secure, individual data repository to store personal data. Access Control Lists with specific constraints (filters based on bound, precision and sampling frequency) are used to control access to the data. Our proposal adds control at the data-collection level, so users can select which data is stored in the Vault. The category-based model generalises the ACL-with-filters model proposed in [17].

7 CONCLUSION

We have presented a data model that integrates data collection and data sharing features. The model is implemented in Privasee, a tool based on the DataBank cloud-IoT architecture, which provides a solution for individual users or companies to manage data in a privacy-preserving way. Privasee suggests default privacy policies by asking the user a small number of questions about their privacy preferences. Data is kept secure in a repository where only the user and authorised services can access it. We plan to improve the usability and experimental evaluations of the system in future work.

REFERENCES

[1] A. Alshehri and R. Sandhu. 2016. Access Control Models for Cloud-Enabled Internet of Things: A Proposed Architecture and Research Agenda. In *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE Press, USA, 530–538.

[2] A. Alshehri and R. Sandhu. 2017. Access Control Models for Virtual Object Communication in Cloud-Enabled IoT. In *IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE Press, USA, 16–25.

[3] S. Alves and M. Fernández. 2017. A graph-based framework for the analysis of access control policies. *Theor. Comput. Sci.* 685 (2017), 3–22.

[4] S. Barker. 2009. The Next 700 Access Control Models or a Unifying Meta-model?. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT '09)*. ACM, New York, NY, USA, 187–196.

[5] E. Bertino, C. Brodie, S. B. Calo, L. F. Cranor, C. Karat, J. Karat, N. Li, D. Lin, J. Lobo, Q. Ni, P. R. Rao, and X. Wang. 2009. Analysis of privacy and security policies. *IBM Journal of Research and Development* 53, 2 (March 2009), 3:1–3:18.

[6] C. Bertolissi and M. Fernández. 2014. A metamodel of access control for distributed environments: Applications and properties. *Information and Computation* 238 (2014), 187 – 207.

[7] M. Blaze, G. Bleumer, and M. Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology-EUROCRYPT'98*. Vol. 1403. Springer, Berlin, 127–144. https://doi.org/10.1007/BFb0054122

[8] A. Botta, W. de Donato, V. Persico, and A. Pescapè. 2016. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems* 56 (2016), 684–700. https://doi.org/10.1016/j.future.2015.09.021

[9] C. Diaz, C. Troncoso, and A. Serjantov. 2008. On the Impact of Social Network Profiling on Anonymity. In *Privacy Enhancing Technologies*, N. Borisov and I. Goldberg (Eds.). Springer Berlin Heidelberg, 44–62.

[10] C. Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener (Eds.). Springer Berlin Heidelberg, 1–12.

[11] M. Fernández, J. Jaimunk, and B. Thuraisingham. 2018. Graph-Based Data-Collection Policies for the Internet of Things. In *Proceedings of the 4th Annual Industrial Control System Security Workshop (ICSS'18)*. ACM, New York, NY, USA, 9–16. https://doi.org/10.1145/3295453.3295455

[12] M. Fernández, J. Jaimunk, and B. Thuraisingham. 2019. Privacy-Preserving Architecture for Cloud-IoT Platforms. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE Press, USA, 11–19. https://doi.org/10.1109/ICWS.2019.00015

[13] M. Fernández, M. Kantarcioglu, and B. Thuraisingham. 2016. A Framework for Secure Data Collection and Management for Internet of Things. In *ICSS '16: Proceedings of the 2nd Annual Industrial Control System Security Workshop*. ACM, Los Angeles, CA, USA, 30–37.

[14] M. Fernández, I. Mackie, and B. M. Thuraisingham. 2019. Specification and Analysis of ABAC Policies via the Category-based Metamodel. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019*, G-J. Ahn, B. M. Thuraisingham, M. Kantarcioglu, and R. Krishnan (Eds.). ACM, 173–184.

[15] H. Haddadi, H. Howard, A. Chaudhry, J. Crowcroft, A. Madhavapeddy, and R. Mortier. 2015. Personal Data: Thinking Inside the Box. *CoRR, ArXiv e-prints* (2015). arXiv:cs.CY/1501.04737

[16] C. T. Hu, D. F. Ferraiolo, D. R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. 2014. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *Special Publication (NIST SP) - 800-162* (jan 2014). https://doi.org/10.6028/NIST.SP.800-162

[17] M. Mun, S. Hao, N. Mishra, K. Shilton, J. Burke, D. Estrin, M. Hansen, and R. Govindan. 2010. Personal Data Vaults: A Locus of Control for Personal Data Streams. In *Proceedings of the 6th International Conference (Co-NEXT '10)*. ACM, New York, NY, USA, Article 17, 12 pages.

[18] D. Núñez. 2016. *New Security Definitions, Constructions and Applications of Proxy Re-Encryption*. Ph.D. Dissertation. University of Malaga, Spain.

[19] D. Núñez. 2018. Umbral, A Threshold Proxy Re-Encryption Scheme. NuCypher Inc and NICS Lab, University of Malaga, Spain.

[20] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma. 2012. Cloud computing for Internet of Things & sensing based applications. In *Sixth International Conference on Sensing Technology (ICST)*. 374–380. https://doi.org/10.1109/ICSensT.2012.6461705

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (Feb. 1996), 38–47. https://doi.org/10.1109/2.485845

[22] L. Sweeny. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570. https://doi.org/10.1142/S0218488502001648

[23] B. Thuraisingham, M. Kantarcioglu, E. Bertino, J. Z. Bakdash, and M. Fernández. 2018. Towards a Privacy-Aware Qunatified Self Data Management Framework. In *Proceedings of the 23rd ACM Symposium on Access Control Models and Technologies (SACMAT'18)*. ACM, New York, NY, USA, 173–184.